ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

TITULACIÓN:

INGENIERO TÉCNICO EN INFORMÁTICA DE GESTIÓN

TÍTULO DEL PROYECTO:

INTERFACES AMIGABLES PARA LA GESTIÓN DE PROCESOS

Alumna: Leticia Osacar Landa

Tutor: Jesús Villadangos Alonso

Pamplona, Septiembre de 2010

Quisiera agradecer, en primer lugar, a mi tutor Jesús, el haberme ofrecido este proyecto y su ayuda a la hora de resolverme dudas y el apoyo al desarrollar este trabajo.

Por supuesto, no puedo olvidarme de agradecer a mis padres y mi hermano el haberme dado la oportunidad de estudiar una carrera, todo su apoyo durante estos años y el haberme ayudado a tirar para adelante en los momentos más duros, no sólo durante el proyecto, si no también durante toda la carrera. Gracias.

También quiero tener unas palabras de agradecimiento para mis amigas Leyre, Claudia y Adriana por darme fuerzas y estar siempre ahí. Y también a mis amigos Dani, David y María por ser un apoyo importantísimo durante todos estos años.

ÍNDICE

ÍNDICE DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

1.1 Introducción

El objetivo principal de este proyecto se centra en el desarrollo de una interfaz gráfica intuitiva para el usuario, mediante la cual se gestionarán diferentes tipos de procesos y que veremos plasmados en una pantalla.

¿Qué es una interfaz gráfica?
Podemos decir que es el conjunto de elementos gráficos que permiten la interacción entre un usuario y una aplicación informática. Existen dos tipos de interfaces:

- Interfaz de línea de comandos.
- Interfaz gráfica de usuario (GUI).

Nosotros nos centraremos en la visualización gráfica de la información.

Las GUIs emplean ventanas para organizar archivos y aplicaciones representadas por iconos y menús. El usuario manipula directamente estos objetos visuales en el monitor señalándolos, seleccionándolos y arrastrándolos o moviéndolos mediante la ayuda del ratón. [1]

Con la palabra interfaz definimos que mediante la utilización de un conjunto de imágenes y objetos gráficos representaremos una información específica y diferentes acciones disponibles en la interfaz.
Este hecho es importante ya que lo que el usuario necesita es que la información que pueda tener almacenada en un momento dado en un sistema de almacenamiento pueda verse de manera clara en una pantalla y así poder verla representada de manera física, ya que la programación a nivel bajo no esta al alcance de todo el mundo.

Gracias a una aplicación de este tipo se puede conseguir la fácil comprensión de los datos recibidos o almacenados en un equipo de trabajo. De esta manera, si se crea una interfaz visualmente atractiva, el usuario no necesitará realizar un gran esfuerzo para entenderla, ya que con sólo echar un vistazo se pueden entender los contenidos de la misma. Entre

otras cosas, se ganaría rapidez a la hora de trabajar, y el sencillo manejo de la aplicación no requeriría una formación compleja anterior a su utilización.
La implementación que hay detrás de la interfaz realiza todo el trabajo costoso y el usuario no debe hacer nada mas que seguir los pasos que se le van marcando según van apareciendo los menús en su pantalla, ya que la navegación no resulta complicada debido a su sencillez.

Para todo esto, el proyecto ha sido estructurado en 3 etapas:

ETAPA 1: Dado que parte de las herramientas que hemos utilizado para la programación de la aplicación no eran conocidas (WPF, XAML, C#), hemos tenido que leer varios manuales y tutoriales para aprender a manejar las mismas.

ETAPA 2: En esta etapa nos hemos centrado en crear la interfaz gráfica con elementos geométricos para hacerlo más intuitivo y agregarles los efectos de animación. Para todo ello hemos utilizado WPF; XAML y también el lenguaje C# para complementar carencias que aparecían con los anteriores lenguajes.

ETAPA 3: En esta última etapa, ya que hemos tenido que utilizar bases de datos, nos hemos centrado en la creación de la misma y también la creación y en rellenar el documento XML. Después, lo único que nos ha hecho falta ha sido perfeccionar la interfaz y agregarle diversos varios elementos para que quedase más completa.

1.2 Herramientas de programación

Podemos dividir esta aplicación en varias partes.

Para programar la aplicación general hemos utilizado la herramienta Visual Studio 2008 Express Edition para C# sobre el sistema operativo Windows XP.
El lenguaje que hemos utilizado ha sido C#, que es un lenguaje orientado a objetos que ha sido desarrollado y estandarizado por Microsoft y forma parte de la plataforma .NET. Tiene una gran similitud a Java aunque incluye mejoras derivadas de otros lenguajes como C++, Java, Visual Basic o Delphi. [6]

Una de las grandes ventajas de utilizar este programa ha sido que constantemente lanza avisos al programador cuando falla en algo que este escribe, provoca redundancia de atributos etc…y que se adelanta a la escritura de las palabras ofreciéndote la opción de no tener que escribir la palabra entera y dejar que el programa lo ponga solo.

En segundo lugar, para diseñar la parte de la interfaz, nos hemos basado en la programación con el lenguaje XAML y WPF.
XAML (eXtensible Application Markup Language) es el lenguaje en el que se basa el WPF (Windows Presentation Foundation) y que soporta clases y métodos de .NET. Es un lenguaje declarativo que, a su vez, se basa en XML de tal manera que nos ofrecen el poder describir interfaces gráficas de usuario ricas desde el punto de vista gráfico. Además, puesto que es un lenguaje que se basa en la utilización de etiquetas para programar lo hace sencillo, intuitivo y fácil de entender. [2]

Windows Presentation Foundation es una tecnología de Microsoft que nos permite el desarrollo de interfaces de interacción tomando las mejores características de las aplicaciones Windows y de las aplicaciones Web. Como gran ventaja tiene que podemos añadir facilidades de interacción para el usuario como: video, animaciones, audio, documentos, navegación o gráficos en 3D. [3]

Al utilizar WPF podemos utilizar recursos más atractivos visualmente para el usuario que si lo programásemos sólo en C# por ejemplo, aunque cabe decir que, desde lo que llamamos “code- behind” tenemos acceso a todas las funciones que se incluyen en WPF, pero nosotros el uso del WPF lo haremos desde la visión del diseñador y mediante XAML. En nuestro caso, por ejemplo, hemos utilizado una animación para crear el efecto zoom en los rectángulos que representan las actividades de un proceso. De la misma manera, también hemos utilizado un efecto de resplandor sobre los mismos rectángulos que cambia de color según el valor de una variable predefinida.

La implementación de las funciones que se ejecutarán durante el uso de la interfaz se han realizado en lenguaje C#. Hemos creado las funciones para conectarse a la base de datos, funciones para crear y rellenar el documento XML, las funciones de navegación de menú

y demás que se llevarán a cabo a la hora de utilizar la aplicación, es decir, la implementación más compleja.

De la misma manera, para la acción de rellenar el documento XML tenemos que conectarnos a una base de datos y acceder a la información contenida en ella, para ello hemos utilizado sentencias en SQL dentro del lenguaje C#.

Hemos requerido utilizado una base de datos local, después, se crean las tablas correspondientes a las entidades que queremos y rellenamos las tablas con los datos que deseemos. En este proyecto la aplicación solamente tomará los datos de la base de datos y por tanto, se dará por hecho que los mismos se encuentran introducidos anteriormente. No será un trabajo que realice nuestra aplicación.

CAPÍTULO 2: REQUISITOS

2.1 Requisitos generales

Lo que trataremos de hacer mediante este proyecto será volcar una información en una interfaz representándola mediante objetos gráficos, de tal manera que haremos que el usuario no tenga que realizar grandes esfuerzos para comprender el significado de la información.

Nuestro programa, al pulsar el botón de iniciar lectura, se conectará con una base de datos local en la cual estarán almacenadas las diferentes informaciones acerca de diferentes procesos, los cuales están compuestos por actividades y a su vez, estas, por diferentes subactividades. A continuación, una vez recogida la información necesaria, se creará un documento XML con la información ordenada de una determinada manera detallada la implementación del programa.
Se ha utilizado el lenguaje XML (Extensible Markup Language) porque ofrece la posibilidad de expresar la información de la manera más estructurada y reutilizable posible y porque no da problemas de compatibilidad entre aplicaciones. [4]

Después, el programa leerá este documento e irá representando en la interfaz las diferentes entidades mediante rectángulos y su información específica también aparecerá plasmada en la ventana. Todo esto se podrá manejar mediante diferentes menús de navegación que estarán implementados.

2.2 Requisitos del sistema y hardware

Para realizar el programa hemos utilizado el Visual Studio 2008 Express Service Pack 1 con el .NET Framework 3.5 SP1, sobre el sistema operativo Windows XP, aunque este programa esta también soportado por los siguientes sistemas operativos: Windows Server 2003, Windows Server 2008, Windows Vista.

Así mismo, también se requiere los siguientes requisitos mínimos del sistema: CPU a 1,6 GHz, 384 MB de RAM, pantalla de 1024 x 768 y disco duro de 5400 rpm. [5]

No debemos olvidarnos que al estar utilizando bases de datos, hemos trabajado con Microsoft SQL Server Compact 3.5.

2.2.1 Librerías necesarias

Tenemos que tener en cuenta que para que la aplicación funcione bien, dado que se utilizan un gran número de métodos durante la implementación, también tenemos que tener cargadas en el programa VS2008, a la hora de programar, las librerías o referencias externas, es decir, tenemos que ir añadiéndolas al programa a medida que vamos confeccionando la aplicación, que en muchas ocasiones, el programa nos va advirtiendo de su necesidad. Las que se explican de aquí en adelante son las que hay que sumar a las que vienen por defecto al crear un nuevo proyecto WPF.

- **System.Windows:** provee de una cantidad considerable de clases elementales de Windows Presentation Foundation necesarias para nuestra aplicación.

- **System.Windows.Controls:** nos da clases para crear elementos, conocidos como controles, de tal manera que hagan posible la interacción del usuario y la aplicación durante el funcionamiento de la misma.

- **System.Windows.Documents:** este espacio de nombres que soportan la creación de documentos de tipo: FixedDocument, FlowDocument y XML Paper Specification (XPS).

- **System.Windows.Input:** proporciona tipos para soportar en sistema de entrada de WPF. Incluye clases para la abstracción de periféricos como el ratón, teclado, ...

- **System.Windows.Media:** provee de elementos que permiten la integración de contenido como dibujos, texto, audio/video ...

- **System.Windows.Media.Imaging:** la necesitamos para codificar y decodificar imagines de tipo bitmap.

- **System.Windows.Media.Animation:** gracias a este espacio de nombres podremos adquirir las funcionalidades de las animaciones tales como los timelines, storyboards y key-frames, utilizados en nuestro caso por ejemplo, para crear el efecto zoom.

- **System.Windows.Navigation:** nos facilita los tipos que nos permiten realizar las navegaciones entre ventanas.

- **System.Windows.Shapes:** nos ofrece el acceso a una librería de formas que pueden ser utilizados en el lenguaje XAML, tales como: elipses, polígonos, rectángulos, etc...

- **System.Xml:** proporciona los estándares necesarios para soportar el procesamiento de documentos XML.

- **System.Xml.Xpath:** contiene el parseador de Xpath necesario para que la aplicación pueda leer y extraer contenidos de documentos XML.

- **System.Windows.Media.Effects:** nos sirve para aplicar efectos visuales las imagines de tipo bitmap.

- **System.Data.SqlServerCe:** este espacio de nombres es el proveedor de datos .NET Compact Framework para SQL Server Mobile. De esta manera podemos crear y administrar bases de datos de SQL Server Mobile en un dispositivo inteligente, así como establecer conexiones con este tipo de bases de datos.

2.3 Diagrama de casos de uso

2.3.1 Caso de uso 1

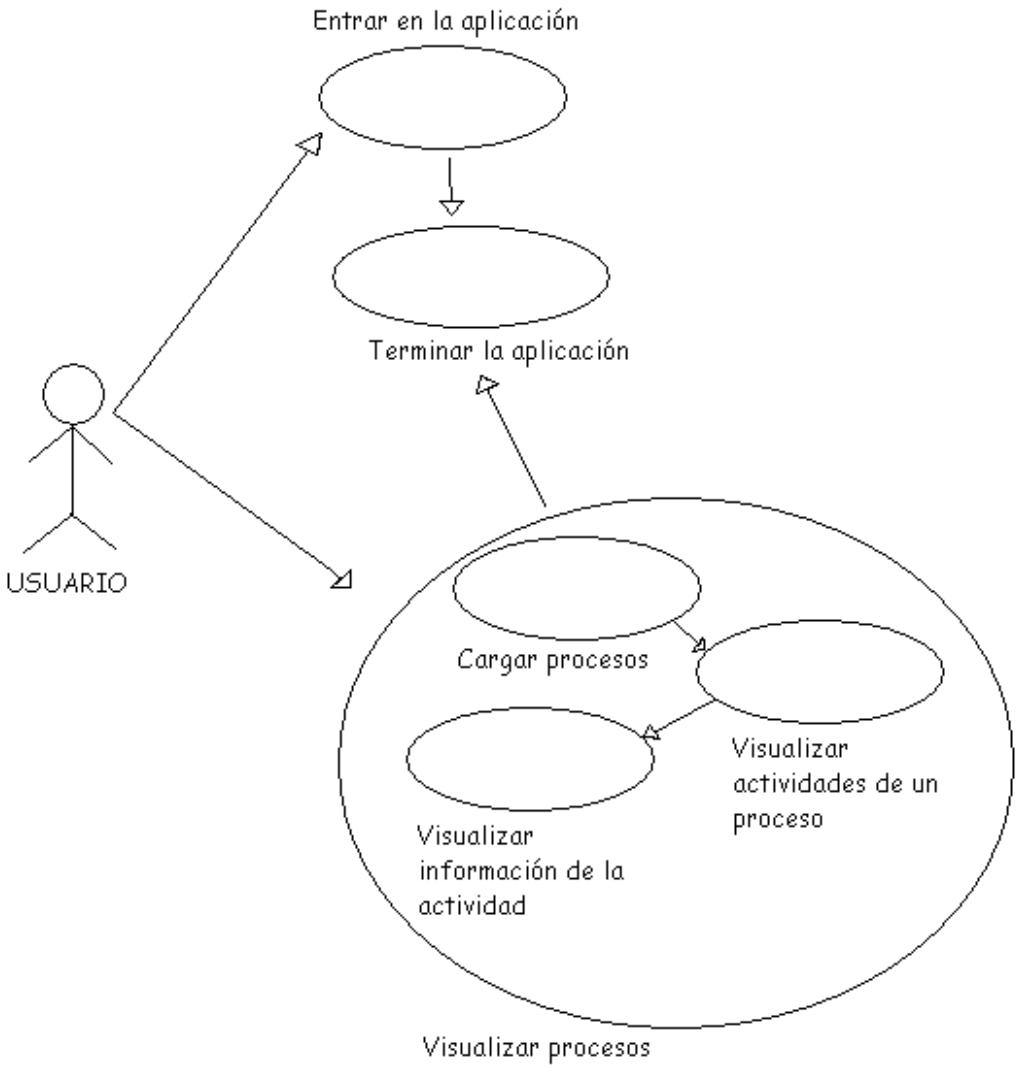


Figura 1 Diagrama de casos de uso 1

- Actores: nuestro actor será el usuario de la aplicación.
- Descripción: el usuario recibe la información almacenada en un dispositivo por medio de una interfaz. En ella aparecerán objetos gráficos y la información escrita que elija.
    - Entrar en la aplicación: el usuario accede a la aplicación.

- Terminar la aplicación: el usuario finaliza la ejecución de la aplicación.
- Visualizar procesos: este caso de uso se compone de varios diferentes:
  - Cargar procesos: Al realizar la conexión con la base de datos y crear y rellenar el documento XML se cargan los procesos existentes en un combobox para que el usuario elija el que quiere visualizar.
  - Visualizar actividades de un proceso: al elegir el proceso se dibujarán en la interfaz los rectángulos correspondientes a las actividades de las que está compuesta el proceso seleccionado.
  - Visualizar información de la actividad: Al pulsar sobre el rectángulo el usuario podrá ver en la interfaz la información relativa a la actividad y las subactividades de las que se compone.

CAPÍTULO 3: ANÁLISIS Y DISEÑO

3.1 Diagrama de secuencia

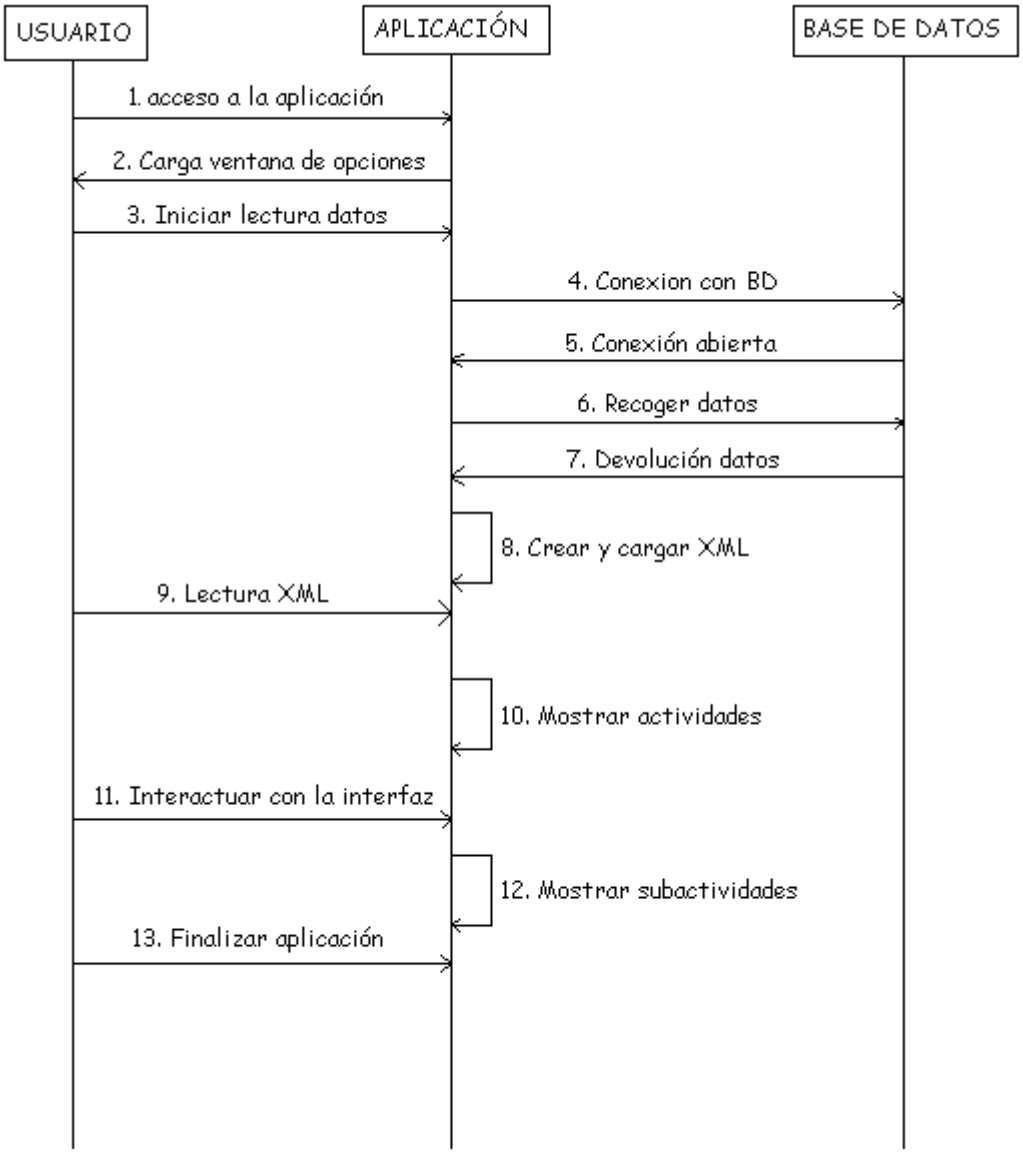


Figura 2 Diagrama de secuencia

3.2 Diagrama de clases y código de la aplicación

3.2.1 Diagrama de clases

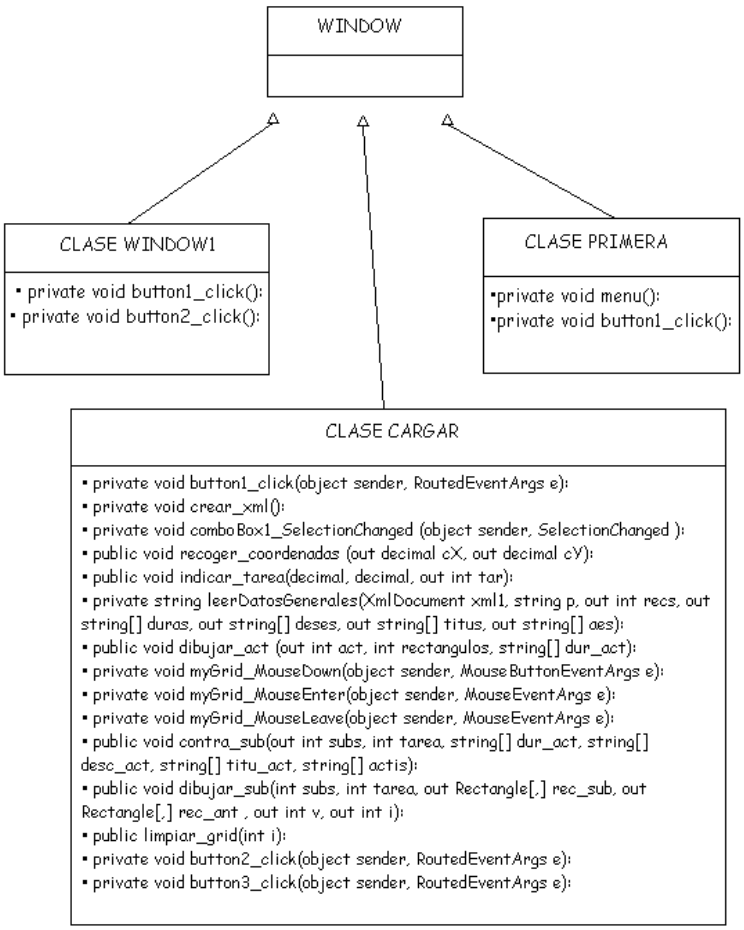


Figura 3 Diagrama de clases

3.2.2 Clases y métodos de la aplicación

A continuación describiremos las funciones que participan en el programa y su funcionamiento:

1. Clase “Window1”:

- *private void button1_click():*

Con esta función al pulsar el botón de "Aceptar" de la ventana se nos creará una nueva ventana, que será la que nos inicie en el programa.

- *private void button2_click():*

Al pulsar el botón de "Cancelar" nos aparecerá en pantalla un *MessageBox* que nos dará la opción de volver al programa o salir de la aplicación.

2. Clase "Primera":

- *private void menu():*

Esta función nos chequea que *RadioButton* esta seleccionado, y dependiendo de cual este marcado se darán diferentes eventos.

- *private void button1_click():*

Al pulsar el botón de "Salir" salimos automáticamente de la aplicación.

3. Clase "Cargar":

- *private void button1_click(object sender, RoutedEventArgs e):*

Dentro de esta función, en primer lugar, llamamos a la función que más adelante explicaremos, crea_xml(). Después, desactivaremos la visibilidad del botón que hemos pulsado y escribiremos en un *Label* que anteriormente hemos inicializado, un mensaje para el usuario.

- *private void crear_xml():*

Esta función lo que hará será crear el documento XML y rellenarlo con la información que se desee de la base de datos. Para ello utilizaremos los objetos de la clase *SqlCeConnection*, para crear la conexión y *XmlTextWriter* para crear y escribir en el documento XML. A continuación, con la ayuda de las sentencias SQL obtendremos la informacion de la base de datos y la insertaremos en los lugares del XML que deseemos.

```
conn = new SqlCeConnection("Data Source ='C:/…/interfaz.sdf';
Password = 'interfaz'");
```

El nombre y la contraseña variarán según lo que se ponga cuando se cree la base de datos. Solamente tenemos que ser correctos y siempre poner la misma.

```
XmlTextWriter myXmlTextWriter = new
XmlTextWriter("C:/…,System.Text.Encoding.UTF8);
```

El significado de UTF8 es que utiliza un sistema de codificación de caracteres. Es decir, es el formato de codificación interna de los caracteres, 8 bits, que se utiliza en documentos XML. También es normal encontrarse con el UTF16, que serian 16 bits.

- *private void comboBox1_SelectionChanged (object sender, SelectionChanged):*

Cuando tengamos operativo el *ComboBox* que estará relleno de los procesos que ha leído del XML, al realizarse la selección de cualquiera de estos items se disparará este evento. Aquí, cogeremos el index del ítem seleccionado y escribiremos el valor en un *Label* que contendrá "PROCESO x", donde x será el valor extraído del *ComboBox* y que se referirá al número de proceso que ha sido seleccionado. Después, crearemos una consulta para poder saber que actividades corresponden al proceso seleccionado por el usuario y lo guardaremos en un string.

Luego, cargaremos el XML y llamaremos a las funciones leerDatosGenerales y dibujar_act.

Cada vez que cambiemos el contenido del *Combobox* haremos también la limpieza del *grid* donde están para que no se sobrepongan las actividades.

```
myGrid.Children.RemoveRange(9, myGrid.Children.Count);
```

Cabe decir que, la función RemoveRange significa que tiene que borrar unos determinados nodos de un rango a otro. En nuestro caso, hemos indicado que el elemento 9 es por el que comienza a borrar porque sabemos que antes de dibujar los rectángulos tenemos 9 elementos dibujados en el *grid* y lo que queremos es que cada vez que elijamos un proceso se borren sólo los rectángulos y el resto de elementos queden donde estén.

- *public void recoger_coordenadas (out decimal cX, out decimal cY):*

Con esta función conseguiremos obtener las coordenadas en las que se encuentra el cursor del ratón en el momento en el que hagamos clic con él sobre uno de los rectángulos que aparecerán el la interfaz.

- *public void indicar_tarea(decimal, decimal, out int tar):*

Transformaremos los datos que hemos recogido de la función recoger_coordenadas en el nombre de la tarea sobre la que hemos pinchado dependiendo de las coordenadas que hemos obtenido, ya que hemos limitado por coordenadas en el *grid* el área que abarcan las los rectángulos para saber a cual nos estamos refiriendo.

- *private string leerDatosGenerales(XmlDocument xml1, string p, out int recs, out string[] duras, out string[] deses, out string[] titus, out string[] aes):*

Extraeremos del documento XML la información sobre las actividades y subactividades que nos interesen. Las guardaremos en strings para luego poder acceder a esta información desde otras funciones.

```
XmlNodeList listaProcesos = xDoc.SelectNodes("Procesos/Proceso");
```

- *public void dibujar_act (out int act, int rectangulos, string[] dur_act):*

Una vez que tengamos el número exacto de actividades de las que esta compuesto el proceso elegido, dibujaremos los rectángulos para representar gráficamente estas actividades. Los rectángulos están definidos como una matriz de rectángulos, en la que en nuestro caso, solamente tiene una fila y puede tener como máximo de columnas el número de actividades columnas.

```
Rectangle[,] rec;
rec = new Rectangle[1, act];
```

Después, daremos dependiendo de una variable seleccionada (en nuestro caso, la duración de la actividad) un diferente valor del resplandor sobre el rectángulo, para así saber distinguir las actividades realizas de las no realizadas.

```
OuterGlowBitmapEffect outer = new OuterGlowBitmapEffect();
OuterGlowBitmapEffect outer2 = new OuterGlowBitmapEffect();
if (dur < 40)
{
    outer.GlowColor = Colors.Red;
    outer.GlowSize = 15;
    rec[0, j].BitmapEffect = outer;
}
else
{
    outer2.GlowColor = Colors.Green;
    outer2.GlowSize = 15;
    rec[0, j].BitmapEffect = outer2;
}
```

También asignaremos un estilo que hemos definido mediante XAML para que los rectángulos que significan actividades se distingan de las subactividades y se comporten de manera diferente, es decir, por ejemplo, que la animación que reproduce el "efecto zoom" solamente tenga valor en las actividades.

```
rec[0, j].Style = (Style)this.Resources["Estilo tarea"];
```

En el caso de que haya 4 o más actividades nuestra ventana se agradará en función del número de actividades de mas que tenemos. Y no tenemos que olvidar que también tenemos que agrandar el tamaño del *grid* en el que están contenidos. El código que hace posible esto es el siguiente:

```
if (act >= 4)
{
    int poner = act - 4 + 1;
    Ventana_cargar.Width = actual;
    Ventana_cargar.Width = actual + (150 * poner);
    myGrid.Width = grid_actual + (160 * poner);
}
else if (act < 4)
{
    myGrid.Width = grid_actual;
    Ventana_cargar.Width = actual;
}
```

- *private void myGrid_MouseDown(object sender, MouseButtonEventArgs e):*

Hemos delimitado dos *grid* en nuestra interfaz, una para las actividades y otra para las subactividades. Por tanto, al hacer clic el *Grid*, llamaremos a las funciones recoger_coordendas, indicar_tarea y limpiar_grid. Cuando recojamos el valor de la tarea que hemos seleccionado (que será un entero) llamaremos a las funciones contar_sub y dibujar_sub.

- *private void myGrid_MouseEnter(object sender, MouseEventArgs e):*

Lo que queremos obtener con esta función es que cuando el puntero del mouse este sobre un rectángulo los demás desaparezcan para darle mas importancia al seleccionado, así que para ello necesitaremos saber sobre que tarea estamos y hacer desaparecer las demás jugando con el atributo de la visibilidad.

- *private void myGrid_MouseLeave(object sender, MouseEventArgs e):*

Mediante esta función obtendremos el efecto contrario al de la anterior función, es decir haremos aparecer los rectángulos que habían desaparecido cuando el ratón salga de los límites establecidos.

- *public void contar_sub(out int subs, int tarea, string[] dur_act, string[] desc_act, string[] titu_act, string[] actis):*

En esta función volveremos a leer el documento XML que hayamos creado pero esta vez nos centraremos en la subactividades y más concretamente de aquí sacaremos cuantas subactividades corresponden a la actividad seleccionada para luego poder dibujarlas.

También rellenaremos un *label* con la información correspondiente a las actividades y subactividades. Recogeremos esta información de los strings que hemos ido rellenando durante la lectura del XML.

```
XmlNodeList listaProcesos = xDoc.SelectNodes("Procesos/Proceso/Actividad");
```

- *public void dibujar_sub(int subs, int tarea, out Rectangle[,] rec_sub, out Rectangle[,] rec_ant , out int v, out int i):*

Con la información de la anterior función, dibujaremos las subactividades tal y como lo hemos hecho con las actividades, es decir, utilizando una matriz de rectángulos de una fila y x columnas para dibujarlas. También aplicaremos el estilo que le corresponde a las subtareas con la siguiente línea de código.

```
rec sub[v, i].Style = (Style)(this.Resources["Estilo subtarea"]);
```

CODIGO XAML DEL ESTILO:

```
<!--Estilo para que las subtareas no se agranden cuando se les ponga el
mouse encima-->
<Style TargetType="{x:Type Rectangle}" x:Key="Estilo_subtarea">
 <Style.Triggers>
  <EventTrigger RoutedEvent="Rectangle.MouseEnter" >
   <EventTrigger.Actions>
    <BeginStoryboard>
     <Storyboard Name="sto">
      <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Height)" AccelerationRatio="1">
      </DoubleAnimation>
      <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Width)" AccelerationRatio="1">
      </DoubleAnimation>
     </Storyboard>
    </BeginStoryboard>
   </EventTrigger.Actions>
  </EventTrigger>
  <EventTrigger RoutedEvent="Rectangle.MouseLeave">
   <EventTrigger.Actions>
    <BeginStoryboard>
     <Storyboard>
      <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Height)" AccelerationRatio="1">
      </DoubleAnimation>
      <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Width)" AccelerationRatio="1">
      </DoubleAnimation>
     </Storyboard>
    </BeginStoryboard>
   </EventTrigger.Actions>
  </EventTrigger>
 </Style.Triggers>
</Style>
```

Figura 4 Ejemplo de XAML para definir el estilo de la ventana de la aplicación

- *public limpiar_grid(int i):*

La función limpiar_grid lo que hace es borrar, cada vez que se llame a esta función, los rectángulos que se han ido dibujando anteriormente para que sólo salgan dibujados los que corresponden a la tarea seleccionada en cada momento, ya que sino se irían acumulando y no se distinguirían entre ellos.

- *private void button2_click(object sender, RoutedEventArgs e):*

Pulsando el botón de “Salir” saldremos de la aplicación.

- *private void button3_click(object sender, RoutedEventArgs e):*

Pulsando el botón de “Atrás” se nos volverá a cargar la ventana original para poder volver a cargar los datos.

3.2.3 Código XAML de la interfaz

Como bien hemos dicho, nuestra interfaz se basa tanto en código en C# como XAML. A continuación, detallaremos las líneas de código XAML que hemos utilizado:

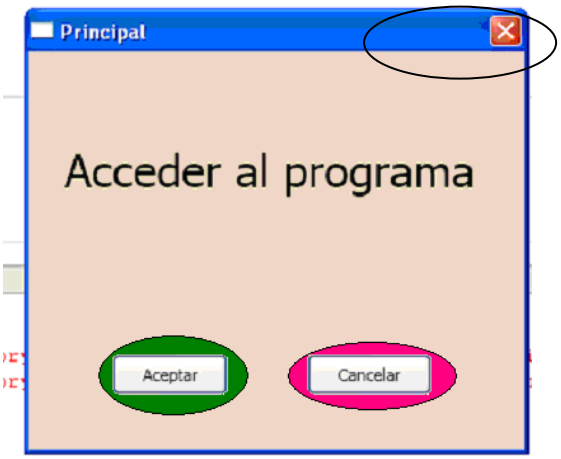


Figura 5 Primera interfaz mostrada al usuario

```
<Window x:Class="menu.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Principal" Height="298" Width="337" Background="PeachPuff"
WindowStartupLocation="CenterScreen" ResizeMode="NoResize">
    <Grid>
        <TextBlock FontSize="30"  Height="47" Margin="24,61,12,0"
Name="textBlock1" VerticalAlignment="Top">Acceder al
programa</TextBlock>
        <Button Margin="56,0,0,37" Name="button1" Height="27"
VerticalAlignment="Bottom" Click="button1_Click"
HorizontalAlignment="Left" Width="77">Aceptar</Button>
        <Button Height="27" HorizontalAlignment="Right"
Margin="0,0,62,37" Name="button2" VerticalAlignment="Bottom"
Width="83" Click="button2_Click">Cancelar</Button>
    </Grid>
```

Figura 6 Código XAML para la creación de la interfaz Figura 4

Como podemos ver, esta sería la primera interfaz que aparecería al ejecutar el programa. Una vez creada la ventana, le indicaremos que cada vez que aparezca este centrada en el centro de la pantalla y bloquearemos las funciones de agrandar y empequeñecer la ventana. Podemos comprobar que los iconos de maximizar y minimizar no están.

Pondremos un *Grid* donde estarán contenidos los elementos que vayamos poniendo, tales como el *TextBlock* que contiene el mensaje "Acceder al programa" y los dos botones de navegación.

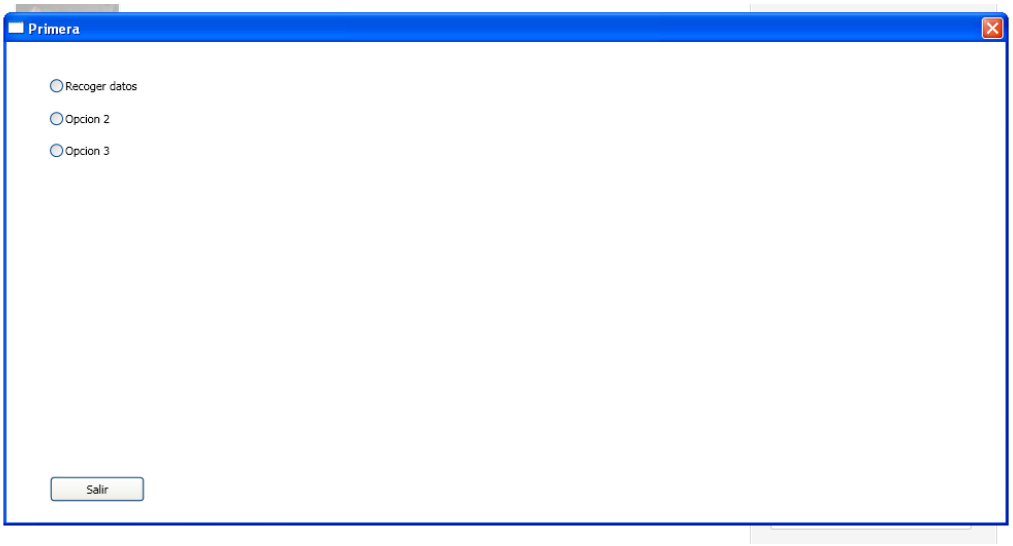


Figura 7 Interfaz para elegir las opciones de navegación

```
<Window x:Class="menu.Primera"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Primera" Height="500" Width="975"
WindowStartupLocation="CenterScreen" ResizeMode="NoResize">
    <Grid>
        <RadioButton Height="15" HorizontalAlignment="Left"
Margin="42,36,0,0" Name="radioButton1" VerticalAlignment="Top" Width="88"
Checked="menu">Recoger datos</RadioButton>
        <RadioButton Height="14" HorizontalAlignment="Left"
Margin="42,68,0,0" Name="radioButton2" VerticalAlignment="Top" Width="64"
Checked="menu">Opcion 2</RadioButton>
        <RadioButton Height="14" HorizontalAlignment="Left"
Margin="42,99,0,0" Name="radioButton3" VerticalAlignment="Top" Width="64"
Checked="menu">Opcion 3</RadioButton>
        <Frame Margin="173,0,0,0" Name="frame1" />
        <Button Height="25" HorizontalAlignment="Left" Margin="42,0,0,20"
Name="button1" VerticalAlignment="Bottom" Width="92" Click="button1_Click"
>Salir</Button>
    </Grid>
</Window>
```

Figura 8 Código XAML para la creación de la interfaz Figura 6

La siguiente ventana que nos aparecerá al dar al botón "Aceptar" será la que se muestra en la Figura 4. Al igual que la primera, las funciones de cambio de tamaño están bloqueadas y aparecerá centra en el centro de la pantalla siempre. Esto será común para todas las ventanas de la aplicación.

Esta compuesta por un *grid* y en el estarán contenidos tres *RadioButton*, un botón y un *frame*, el cual cambiará de apariencia en función del *RadioButton* que este seleccionado.

Un *frame* es un como un contenedor que en nuestro caso no se utiliza pero que puede servir para cargar diferentes páginas (*Page*) o para cargarnos una página web.

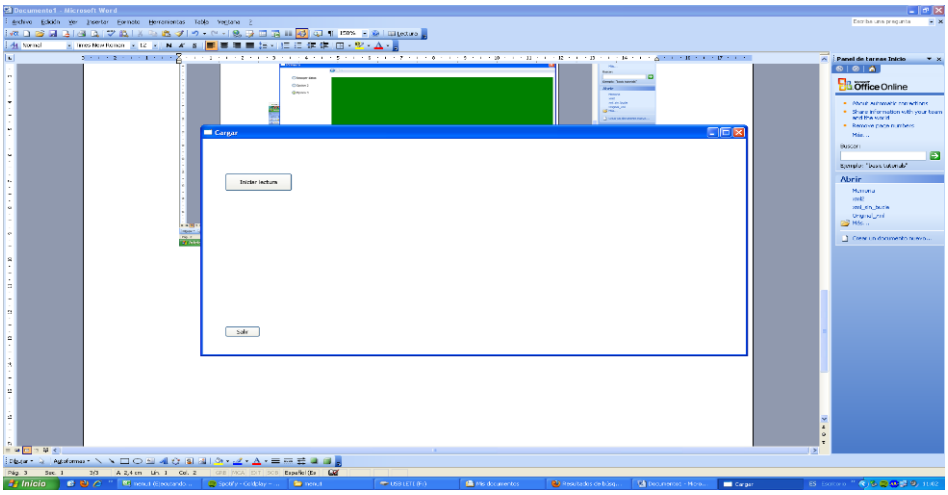


Figura 9 Interfaz para el inicio de la lectura de datos.

```
<Window x:Class="menu.Cargar"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Cargar" Height="500" Width="975" WindowStartupLocation="CenterScreen"
Background="White" Name="Ventana_cargar">
 <Window.Resources>
  <!--Estilos para agrandar las TAREA en plan ZOOM-->
   <Style TargetType="{x:Type Rectangle}" x:Key="Estilo_tarea">
    <Style.Triggers>
     <EventTrigger RoutedEvent="Rectangle.MouseEnter">
      <EventTrigger.Actions>
       <BeginStoryboard>
        <Storyboard>
         <DoubleAnimation To="150" Duration="0:0:01"
Storyboard.TargetProperty="(Height)" AccelerationRatio="1"></DoubleAnimation>
         <DoubleAnimation To="150" Duration="0:0:01"
Storyboard.TargetProperty="(Width)" AccelerationRatio="1"></DoubleAnimation>
        </Storyboard>
       </BeginStoryboard>
      </EventTrigger.Actions>
     </EventTrigger>
     <EventTrigger RoutedEvent="Rectangle.MouseLeave">
      <EventTrigger.Actions>
       <BeginStoryboard>
        <Storyboard>
         <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Height)" AccelerationRatio="1"></DoubleAnimation>
         <DoubleAnimation To="50" Duration="0:0:01"
Storyboard.TargetProperty="(Width)" AccelerationRatio="1"></DoubleAnimation>
        </Storyboard>
       </BeginStoryboard>
      </EventTrigger.Actions>
     </EventTrigger>
    </Style.Triggers>
   </Style>
```

Figura 10 Código XAML para la creación de la ventana y primer estilo de rectángulos.

```
<!--Estilo para que las subtareas no se agranden cuando se les ponga el
mouse encima-->
        <Style TargetType="{x:Type Rectangle}" x:Key="Estilo_subtarea">
            <Style.Triggers>
                <EventTrigger RoutedEvent="Rectangle.MouseEnter" >
                    <EventTrigger.Actions>
                        <BeginStoryboard>
                            <Storyboard Name="sto">
                                <DoubleAnimation To="50"
Duration="0:0:01" Storyboard.TargetProperty="(Height)"
AccelerationRatio="1"></DoubleAnimation>
                                <DoubleAnimation To="50"
Duration="0:0:01" Storyboard.TargetProperty="(Width)"
AccelerationRatio="1"></DoubleAnimation>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger.Actions>
                </EventTrigger>
                <EventTrigger RoutedEvent="Rectangle.MouseLeave">
                    <EventTrigger.Actions>
                        <BeginStoryboard>
                            <Storyboard>
                                <DoubleAnimation To="50"
Duration="0:0:01" Storyboard.TargetProperty="(Height)"
AccelerationRatio="1"></DoubleAnimation>
                                <DoubleAnimation To="50"
Duration="0:0:01" Storyboard.TargetProperty="(Width)"
AccelerationRatio="1"></DoubleAnimation>
                            </Storyboard>
                        </BeginStoryboard>
                    </EventTrigger.Actions>
                </EventTrigger>
            </Style.Triggers>
        </Style>
    </Window.Resources>
```

Figura 11 Código XAML para el segundo estilo de rectángulo.

```
 <Grid Name="myGrid" Height="208" Width="787" HorizontalAlignment="Center"
VerticalAlignment="Top" MouseDown="myGrid_MouseDown" MouseEnter="myGrid_MouseEnter"
MouseLeave="myGrid_MouseLeave">
  <Button Margin="-50,75,0,95" Name="button1" Click="button1_Click"
HorizontalAlignment="Left" Width="119">Iniciar lectura</Button>
   <Grid Height="123" Margin="0,0,0,-220" Name="grid1" VerticalAlignment="Bottom">
   <Label HorizontalAlignment="Left" Margin="109,46,0,47" Name="label5" Width="157"
FontSize="18">SUBACTIVIDADES</Label>
   </Grid>
  <Button Height="23" HorizontalAlignment="Left" Margin="-50,0,0,-220"
Name="button2" VerticalAlignment="Bottom" Width="62"
Click="button2_Click">Salir</Button>
   <Button Height="22" HorizontalAlignment="Left" Margin="-50,0,0,-186"
Name="button3" VerticalAlignment="Bottom" Width="62"
Click="button3_Click">Atrás</Button>
   <ComboBox HorizontalAlignment="Left" Margin="-50,0,0,17" Name="comboBox1"
Width="119" SelectionChanged="comboBox1_SelectionChanged" Height="23"
VerticalAlignment="Bottom" />
   <Label Height="44" Margin="0,12,208,0" Name="label4" VerticalAlignment="Top"
FontSize="24" FontFamily="Verdana" HorizontalAlignment="Right" Width="166"></Label>
   <Label HorizontalAlignment="Left" Margin="-50,0,0,-137" Name="label1"
Width="189" Height="148" VerticalAlignment="Bottom"></Label>
   <Label Height="28" HorizontalAlignment="Left" Margin="-50,0,0,46" Name="label2"
VerticalAlignment="Bottom" Width="276" FontSize="13"></Label>
   <Label HorizontalAlignment="Left" Margin="131,0,0,72" Name="label3" Width="121"
FontSize="18" Height="28" VerticalAlignment="Bottom">ACTIVIDADES</Label>
 </Grid>
</Window>
```

Figura 12 Código XAML para la creación de los objetos de la interfaz.

Esta última figura (Figura5) sería en la que trabajaríamos la parte de la aparición de los datos que el usuario ha pedido.

La estructura en la que se basa es igual que las anteriores. Tenemos un *grid* en el cual iremos poniendo los elementos tales como botones, *labels*, *ComboBox* etc…Muchos de estos elementos, como los *labels* y el *ComboBox* que no aparecen visibles en un principio pero que a medida que ocurren las funciones y acciones programadas aparecen.

Lo más destacable de esta interfaz es la animación que afecta a los rectángulos. Como hemos explicado en anteriores apartados, los rectángulos que representan las actividades tiene una animación efecto zoom que se realiza con *eventTriggers* que provocan dos tipos de animaciones diferentes: una realiza el aumento de tamaño y el otro la disminución hasta el valor original del rectángulo.

Tenemos dos tipos de estilos que realizan animaciones. Una no agradará y empequeñecera el tamaño y el referido a las subactividades dejara el tamaño del rectángulo tal y como es, es decir, no cambia el tamaño.

3.2.4. Estructura del XML

El código XML que esta en el recuadro es el esquema que sigue en esta aplicación.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Procesos>
  <Proceso>
    <Id_proceso></Id_proceso>
    <Actividad>
      <Id_act></Id_act>
      <Duracion></Duracion>
      <Descripcion></Descripcion>
      <Titulo></Titulo>
      <Subactividades>
        <Subactividad>
        </Subactividad>
      </Subactividades>
    </Actividad>
  </Proceso>
</Procesos>
```

Figura 13 Estructura del código XML.

Tenemos que tener la información bien estructurada, es decir las partes de las que se compone estén bien definidas, y estas a su vez de otras. Debemos tener claro que los documentos XML sólo deben tener un nodo raíz del que se desarrollan todos los demás.

Las partes quedarán definidas mediante etiquetas, las cuales son marcas hechas en el documento que marcarán las porciones como elementos. Y tampoco nos debemos olvidar de colocar la cabecera del documento siempre. (Nuestra función crear_xml la hace automáticamente)

Como sabemos, la etiqueta procesos engloba todos los procesos que existen almacenados. De esta manera almacenaremos los procesos con su información particular. Dentro de la etiqueta Id_proceso se encontrará el código de cada proceso. A continuación viene la información de las actividades que componen cada proceso. Dentro de ellas tendremos el código de cada actividad (Id_act) la duracion de cada una (duracion), una breve descripción (descripción) y el título.

Como cada actividad se compone de ciertas subactividades, estas también estarán reflejadas, se dispone de una etiqueta (subactividades) que engloba a todas la subtareas de las que se compone, y dentro de ella la etiqueta subactividad contendrá el código de cada una de ellas.

CAPÍTULO 4: BASE DE DATOS Y DISEÑO DE LA INTERFAZ

4.1 Base de datos

Ya que esta aplicación se conecta a una base de datos para poblar el documento XML con los datos referidos a los procesos, que luego van a ser leídos por el programa, hemos creado un esquema entidad- relación para el correcto diseño de la base de datos.

- Esquema entidad-relación:

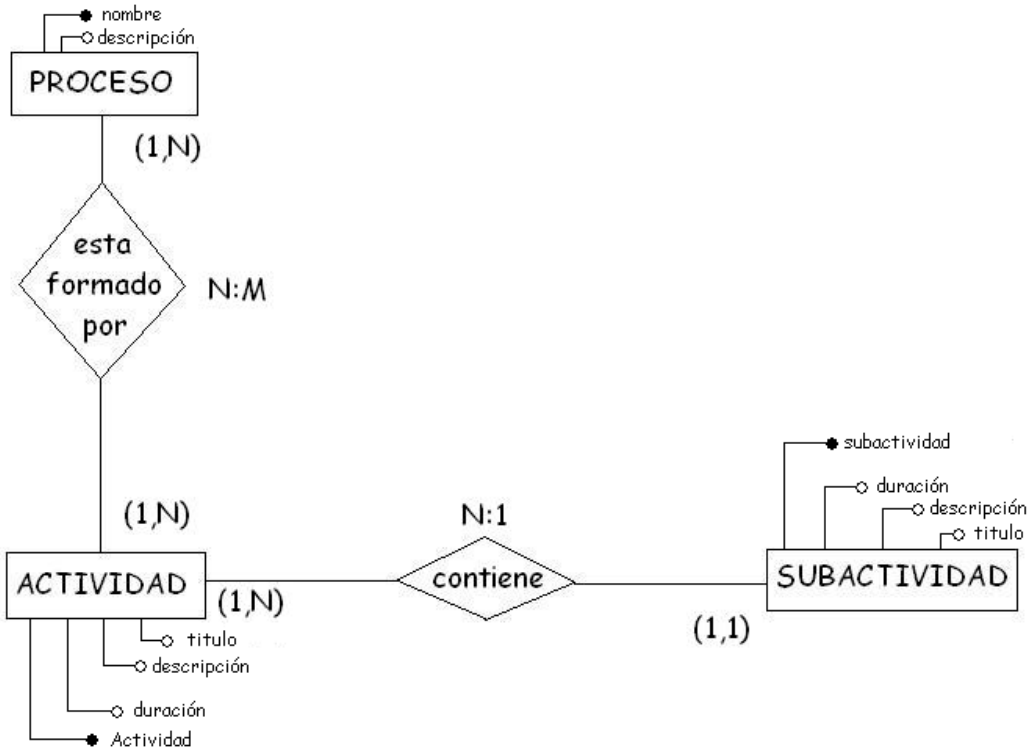


Figura 14 Diagrama entidad/relación para la base de datos.

- Análisis de las entidades y atributos:

PROCESO	ACTIVIDAD	SUBACTIVIDAD
NOMBRE	ACTIVIDAD	COD_SUB
DESCRIPCIÓN	DURACIÓN	DURACIÓN
	DESCRIPCIÓN	DESCRIPCIÓN
	TITULO	TITULO

Figura 15 Tabla con los atributos de cada entidad.

PROCESO: Esta entidad nos represente el proceso del cual queremos obtener la información, el cual esta compuesto por actividades y esta a su vez, de subactividades.
-Atributos:

- Nombre: nombre del proceso.
- Descripción: breve descripción del proceso.

ACTIVIDAD: mediante esta entidad sabemos que actividades esta compuesto el proceso que hemos seleccionado para estudiar.
-Atributos:

- Actividad: Código de la actividad.
- Duración: lo que se tarda en realizar la actividad.
- Descripción: descripción de la actividad.
- Titulo: Nombre de la actividad.

SUBACTIVIDAD: la última entidad nos ofrece la información de las subactividades que componen una actividad.
-Atributos:

- Cod_sub: Código de la subactividad.
- Duración: duración de la subactividad.
- Descripción: descripción de la subactividad.
- Titulo: Nombre de la subactividad.

- Análisis de las relaciones:

esta_formado_por (N: M): Un proceso esta compuesto por una o varias actividades y una actividad puede ser parte de un proceso concreto o de varios diferentes, un proceso no conlleva que tengo unas actividades propias.
-Atributos:

- Nombre: cogeríamos en nombre del de la entidad proceso.
- Actividad: seria la/s actividades correspondientes a cada proceso.

- Paso a tablas:

PK: clave primaria.

FK: clave foránea.

PROCESO:

Nombre	descripción
PK	

Figura 16 Tabla con los atributos de PROCESO.

ACTIVIDAD:

actividad	duración	descripción	título
PK			

Figura 17 Tabla con los atributos de ACTIVIDAD

SUBACTIVIDADES:

cod_sub	duracion	descripción	titulo	cod_act
PK				FK

Figura 18 Tabla con los atributos de SUABCTIVIDADES

Cod_act es una clave foránea ya que corresponde a las actividades de la tabla Actividad. La relación entra ambas seria que a cada actividad le corresponden una/s subactividades concretas y así sabremos cuales corresponden a cual.

ESTA_FORMADO POR:

Nombre	actividad
FK	FK

Figura 19 Tabla con los atributos de ESTA_FORMADO_POR

La composición de esta tabla se hace con 2 claves foráneas, que corresponden a las claves primarias de las tablas Proceso (nombre) y Actividad (actividad).

4.2 Diseño de la interfaz

El diseño de la interfaz esta basado en la sencillez.

Primeramente, cuando el usuario accede a la aplicación, le aparece una ventana que le proporciona dos opciones a realizar: seguir adelante con la aplicación o la de terminar con la aplicación antes de comenzar.

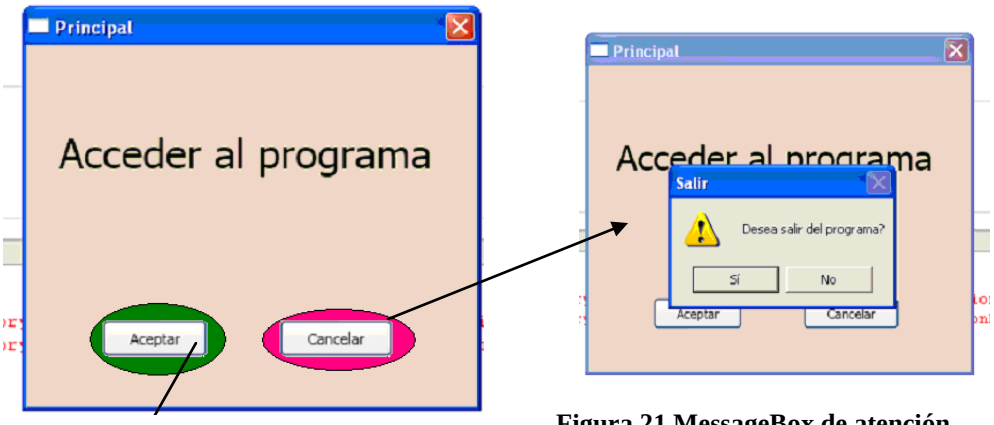


Figura 20 Primera interfaz.

Figura 21 MessageBox de atención.

Una vez pulsado el botón de aceptar, nos aparecerá una nueva pantalla en la que tendremos varias opciones a poder realizar:

- Recoger los datos de los procesos a estudiar.
- Dos opciones de navegación de menús.

A nosotros la que nos interesa es la primera, la de recogida de datos.

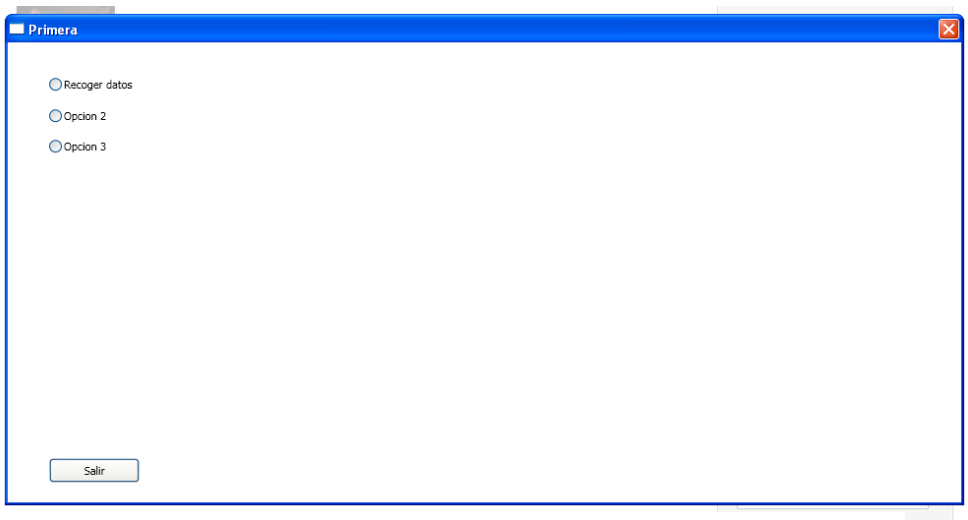


Figura 22 Interfaz para la elección de opciones de navegación.

Al pulsar esta opción se nos cargan nuevos elementos en la ventana. Nos aparecerán dos botones nuevos: el de inicio de lectura y el de salir de la aplicación.

Si pulsamos el de inicio de lectura nos llevará una nueva ventana y si pulsamos el de salir saldremos automáticamente de la aplicación.

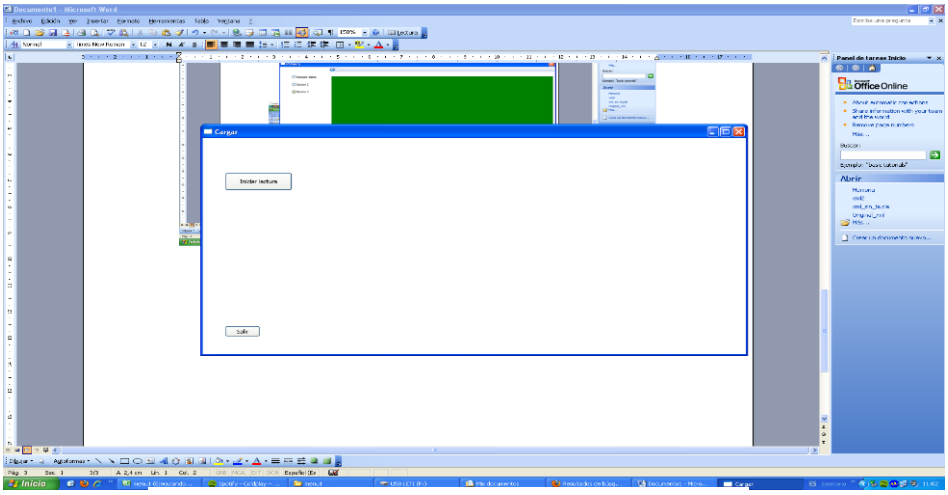


Figura 23 Interfaz de inicio de lectura de los datos.

Una vez pulsado el botón “Iniciar lectura” nos aparecerá la ventana de esta manera:

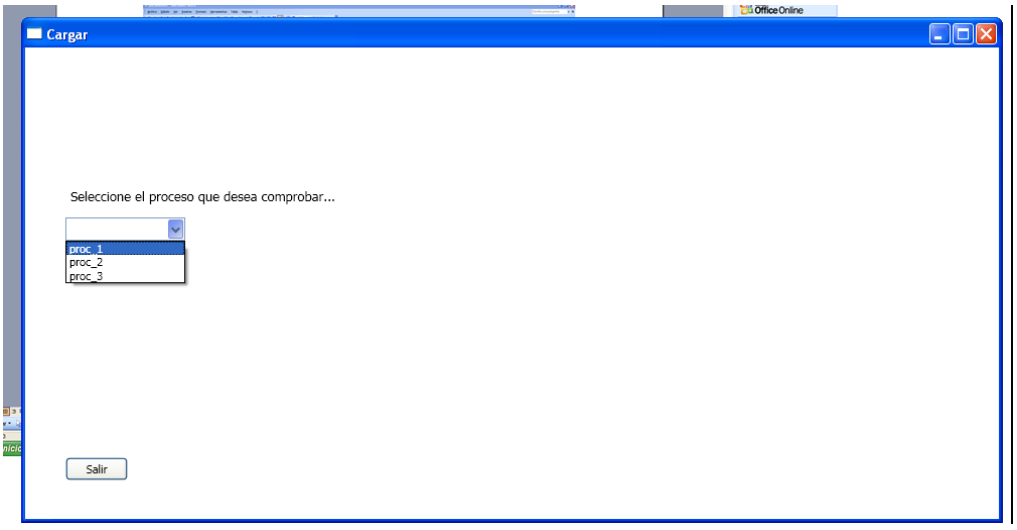


Figura 24 Interfaz para la elección del proceso a estudiar.

Dentro de un panel que se despliega, y después de que la aplicación haya leído la información del XML, nos aparecerán los procesos que existen para analizar. Seleccionamos el que deseemos ver y a continuación se cargarán las actividades

correspondientes al proceso seleccionado. El color del borde cambia en función de una variable que hemos establecido, si es menor que un número concreto aparecerá de color rojo y sino, de verde. El sentido que quiere darse con la diferencia de colores puede entenderse como si la actividad estuviese realizada o sin realizar. Ver un color es más rápido que leer.

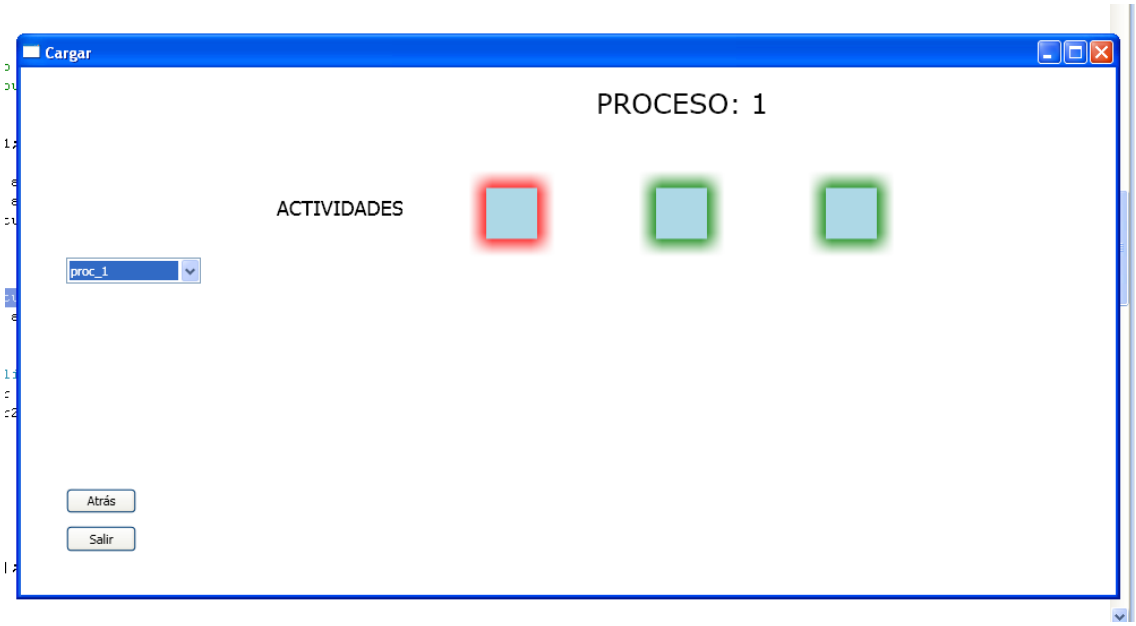


Figura 25 Interfaz una vez cargadas las actividades correspondientes.

Al hacer clic en el rectángulo que queramos nos aparecerá la información correspondiente a la actividad en el lado izquierdo de la interfaz. Si mantenemos el puntero del ratón sobre cada rectángulo sin pinchar en el, este se nos agrandará y hará desaparecer los que no estén seleccionados para darle protagonismo al seleccionado por el usuario. De la misma manera, hacer clic en el rectángulo nos aparecerán las subactividades correspondientes a esa actividad en la parte inferior de la interfaz.

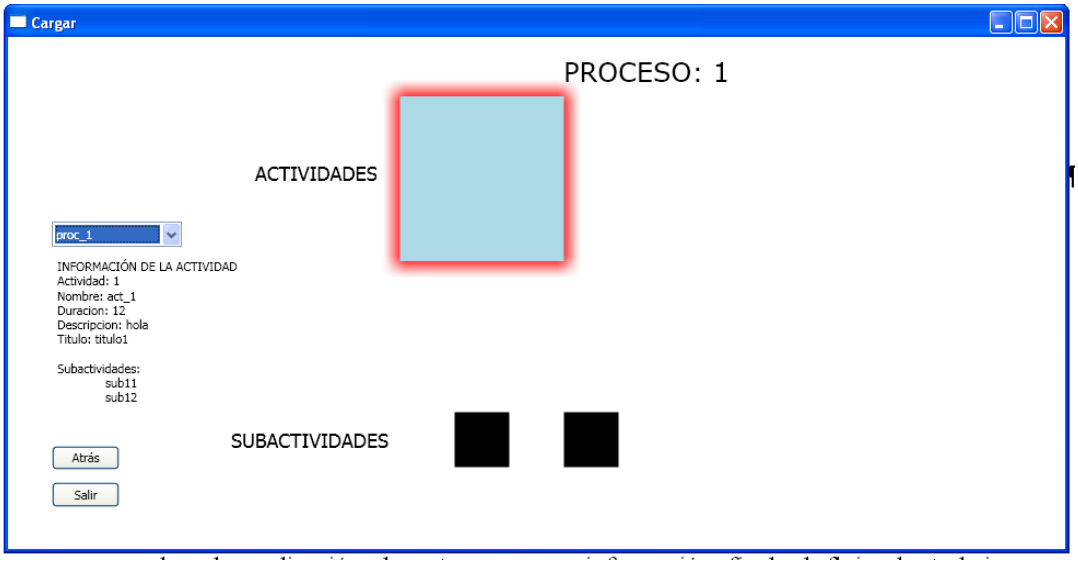


Figura 26 Interfaz una vez pulsado el ratón sobre la actividad

Podemos ver también que el tamaño de la interfaz aumentará si tenemos más de tres actividades que mostrar:

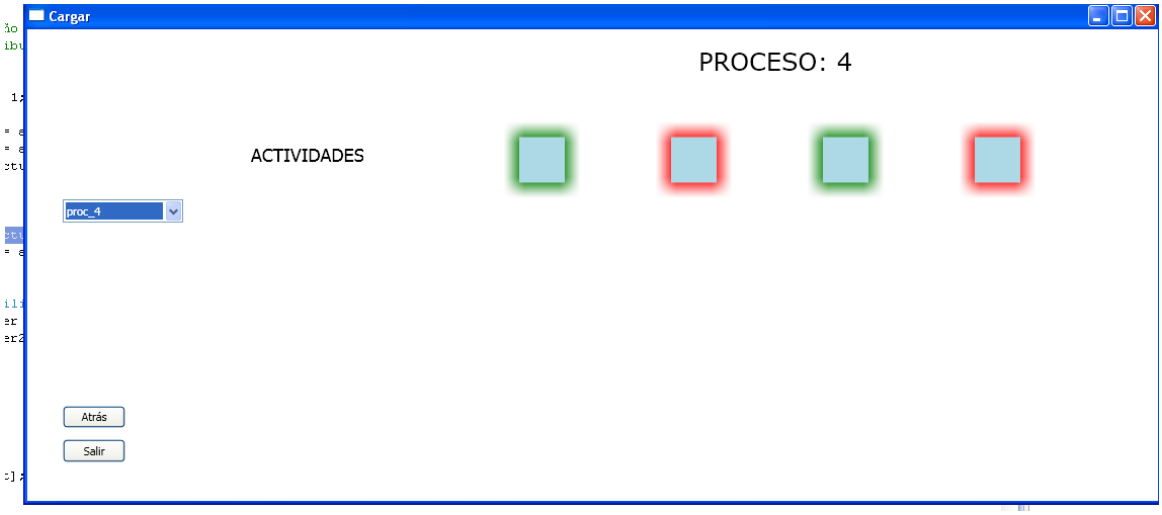


Figura 27 Ejemplo de interfaz con mas de tres actividades.

CAPÍTULO 5: CONCLUSIONES

Después de desarrollar esta aplicación la primera conclusión clara ha sido el conocimiento de un lenguaje nuevo, como era C#. El saber que tenia bastante parecido a otros lenguajes ya aprendidos durante la carrera como Java nos ha ayudado a la hora de entenderlo pero siempre teniendo a mano manuales y tutoriales.
También se han adquirido conocimientos sobre la divino del trabajo en un proyecto y en las diferentes etapas que se pueden desarrollar en el. La importancia de tener esquematizado todo los requisitos que queremos que nuestra aplicación cumpla son de gran ayuda a la hora de organizar el trabajo a realizar.
Nos hemos dado cuenta que a la hora de programar se deben tener las cosas muy claras ya que al programar la ideas se mezclan y pueden llevar a confusiones, mezclas de variables, código innecesario, etc...

Al final, hemos podido intuir de alguna manera como debe realizarse un trabajo, aunque esta intuición sea pequeña, ya que no es un proyecto de grandes dimensiones.

CAPÍTULO 6: LÍNEAS FUTURAS

Dado que este proyecto abarca solamente la parte de recopilación y transformación de datos para el uso del usuario, podríamos extender su funcionalidad y completar la aplicación en forma de un programa con menús de navegación en los que el usuario podría cambiar la apariencia de la aplicación, asignar administradores de la aplicación, proteger la misma con contraseñas, etc...

También se podría añadir una funcionalidad diferente en el caso de que el número de actividades que tengamos sea grande. Es decir, en nuestro caso, la interfaz se va agrandando cuando se sobrepasan las tres actividades. Al ir añadiéndose en horizontal, llegará un momento en el que no entrarán más en la pantalla, así que sería interesante buscar una manera de visualizar todas las actividades en una misma pantalla.

BIBLIOGRAFÍA

[1] Tutoriales de inicio WPF

One World CD. Microsoft University TOUR 09

[1] Windows Presentation Foundation nº7 (Cuadernos técnicos dotnetmania).

Miguel Katrib, Mano de Valle, Iskander Sierra, Yamil Hernandez.

[1] Definición de interfaz

http://www.slideshare.net/ximenatabares/historia-de-la-interfaz-grfica

[2] ¿Qué es Windows Presentation Foundation?

http://es.wikipedia.org/wiki/Windows_Presentation_Foundation

[3] Definición de XAML

http://es.wikipedia.org/wiki/XAML

[4] Definición de XML

http://es.wikipedia.org/wiki/Xml

[5] Requisitos de instalación del VS 2008 Express

http://www.microsoft.com/downloads/details.aspx?displaylang=es&FamilyId=FBEE1648-7106-44A7-9649-6D9F6D58056E#QuickInfoContainer

[6] Historia del lenguaje C#

http://es.wikipedia.org/wiki/C_Sharp

http://www.oscarsotorrio.com/category/XML.aspx

http://msdn.microsoft.com/es-es/library/bb383877.aspx

INTERFACES AMIGABLES PARA LA GESTIÓN DE PROCESOS

Leticia Osácar Landa
Ingeniería Técnica en Informática de Gestión
Septiembre 2010, Pamplona

Introducción

- OBJETIVO DEL PROYECTO:

  Desarrollo de una interfaz intuitiva para el usuario.

- ETAPAS

  1. Primer contacto con las herramientas de programación.
  2. Creación y diseño de la interfaz.
  3. Creación de la base de datos.

Introducción

- ¿Qué es una interfaz gráfica?

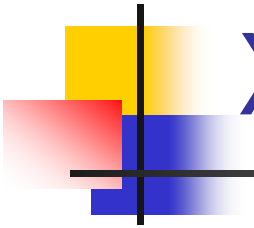
  Conjunto de elementos que permiten la interacción entre un usuario y una aplicación informática.
- Conseguimos plasmar información en una pantalla

Herramientas de programación

- Visual Studio 2008 Express Edition.
- Lenguaje C#.
- WPF (Windows Presentation Foundation).
- XAML.
- XML.
- Sentencias SQL.

WPF (Windows Presentation Foundation)

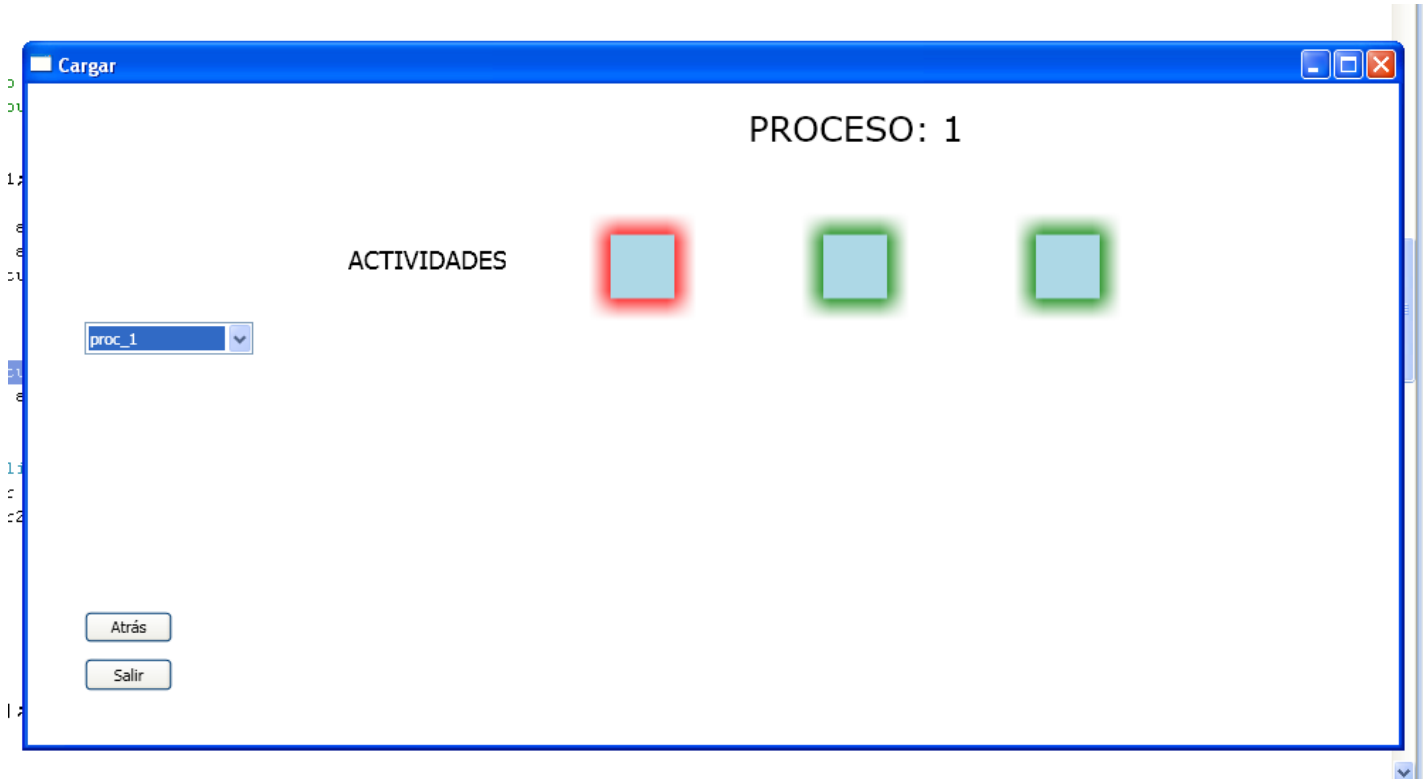
- Tecnología de Microsoft.
- Desarrollo de interfaces.
- Añadir propiedades de interacción:
  - Video.
  - Animaciones.
  - Audio...



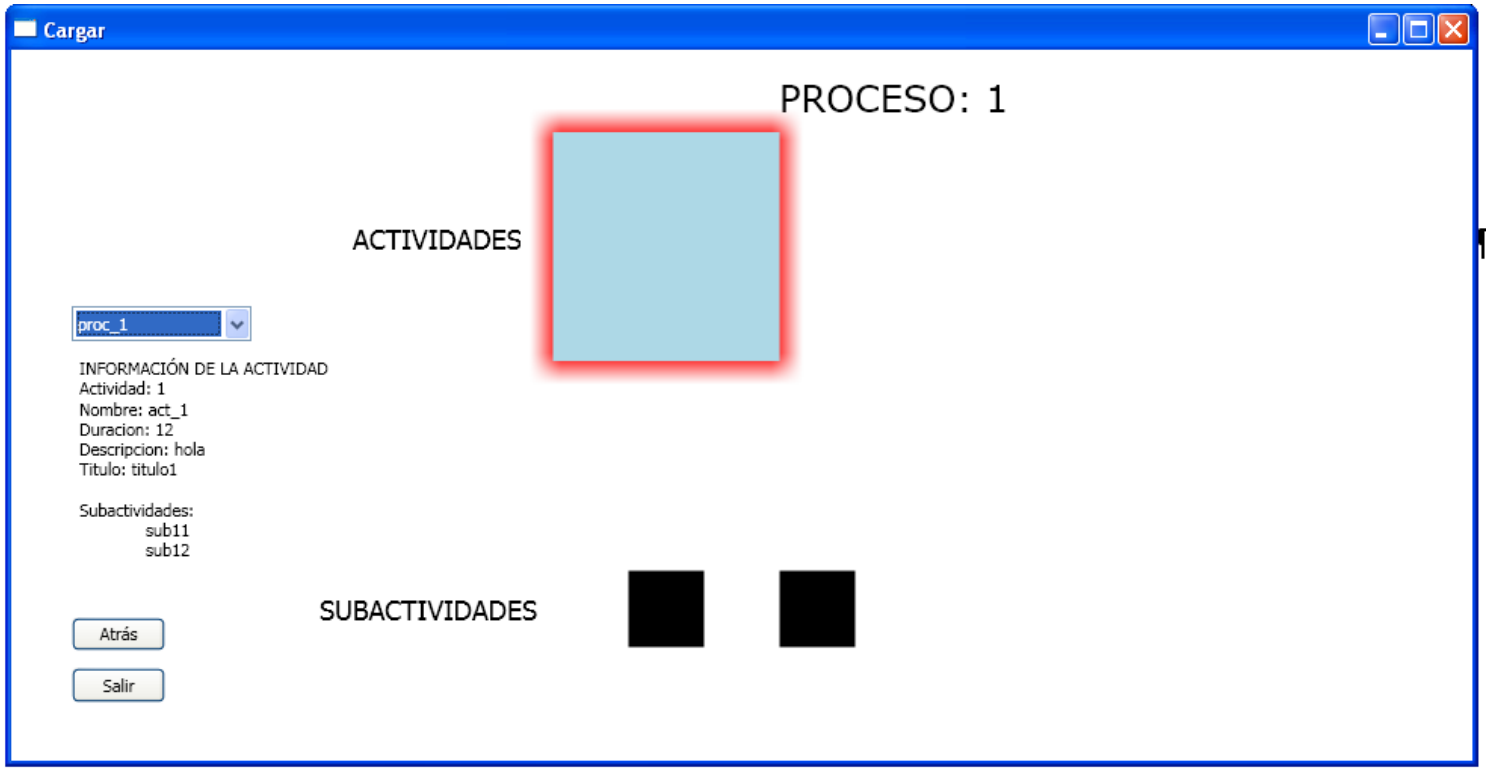
XAML

- Lenguaje en el que se basa WPF.
- Soporta clases y métodos .NET.
- Lenguaje declarativo basado en XML.
- Descripción de interfaces gráficas.
- Etiquetas.

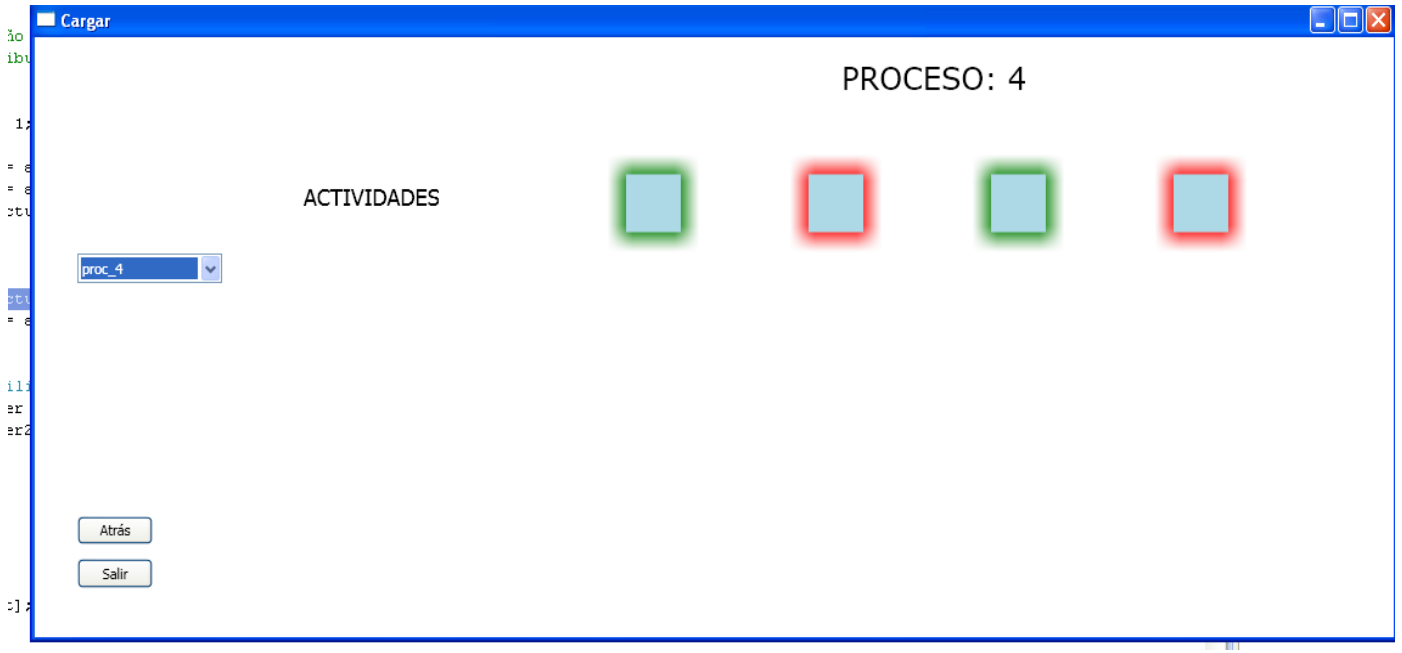
Funcionalidades del sistema - I



Funcionalidades del sistema - II



Funcionalidades del sistema - III



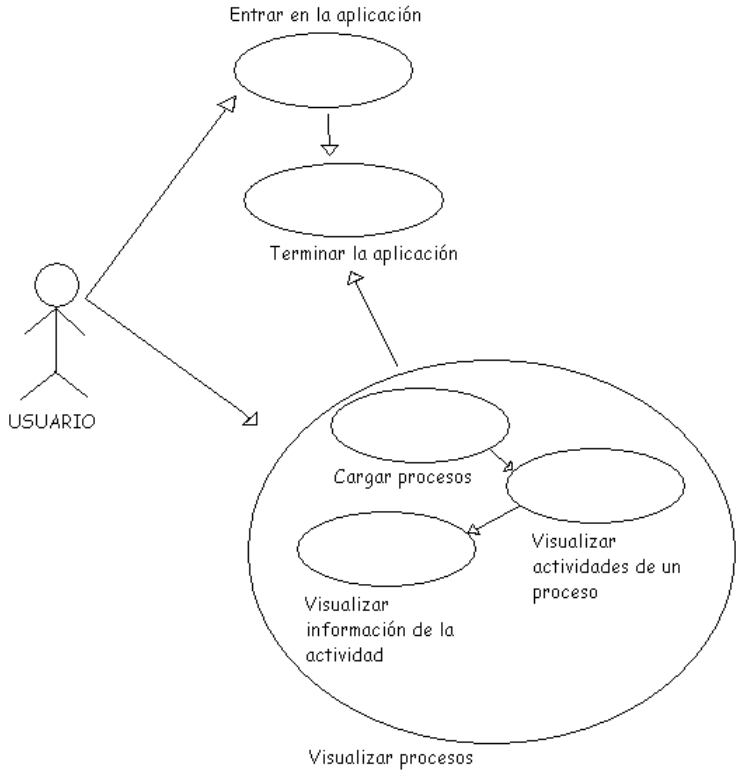
Requisitos del sistema y hardware

- Uso del lenguaje XML para el intercambio de información entre los diferentes módulos que se desarrollen
- Visual Studio 2008 Express Edition Service Pack 1 con el framework 3.5 SP1
- Requerimientos técnicos:

  - ordenador de sobremesa

Diagrama de casos de uso

- Actor: Usuario.
- Descripción:
  - Entrar en aplicación.
  - Terminar aplicación.
  - Visualizar procesos.
    - Cargar procesos.
    - Visualizar actividades.
    - Visualizar información.



Esquema entidad/relación

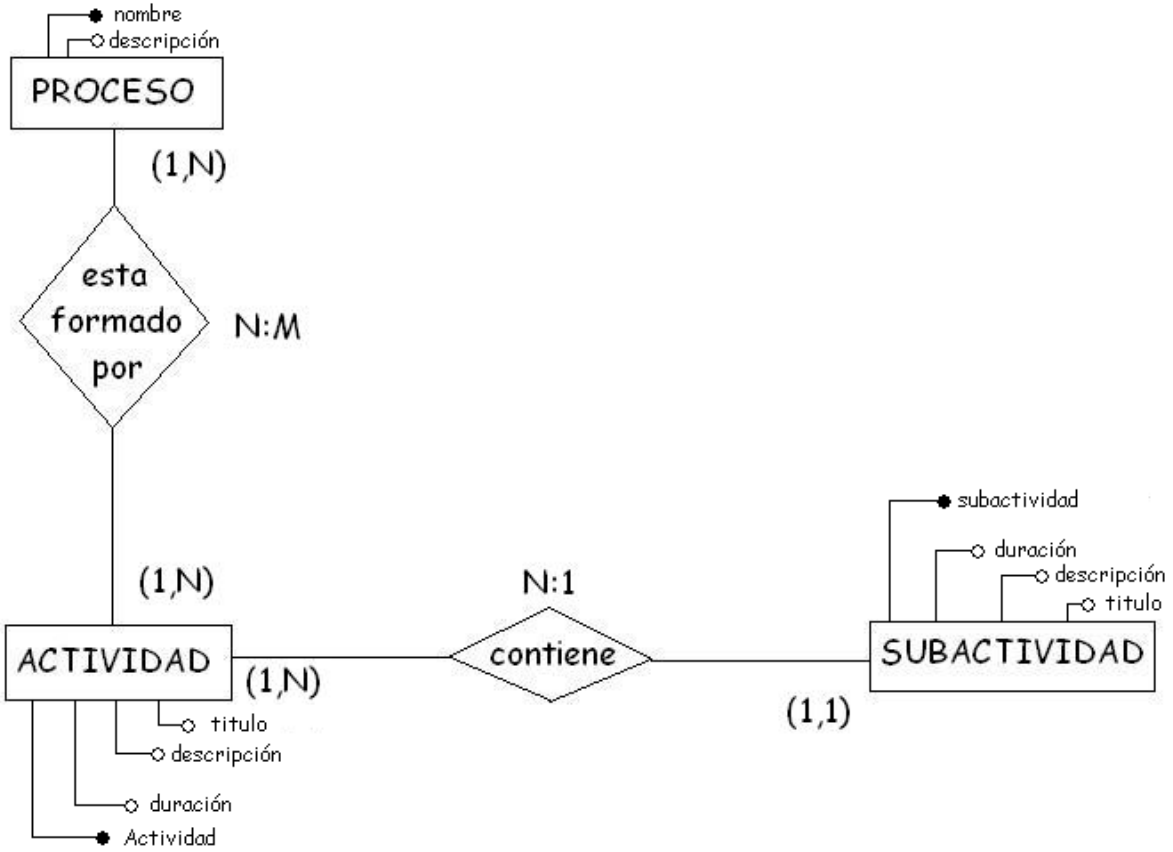


Diagrama de secuencia

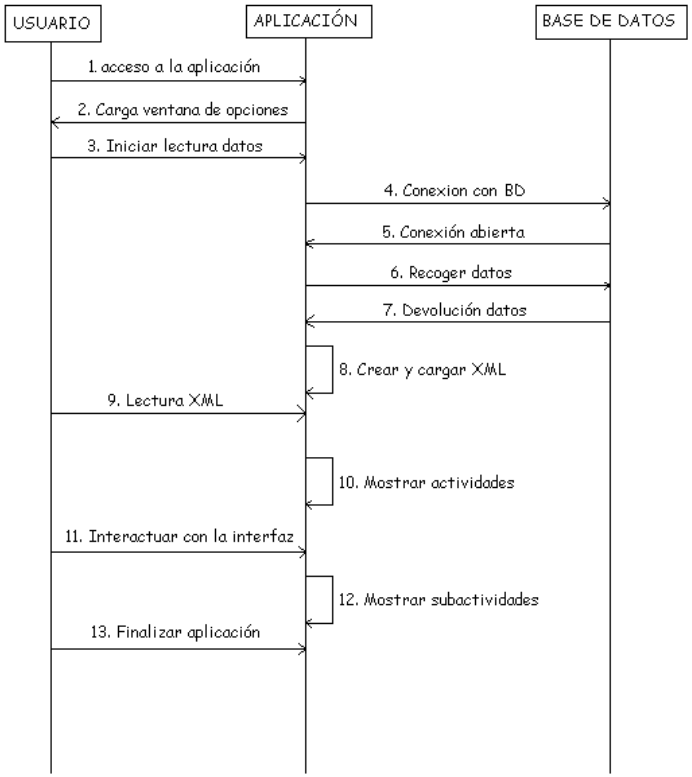
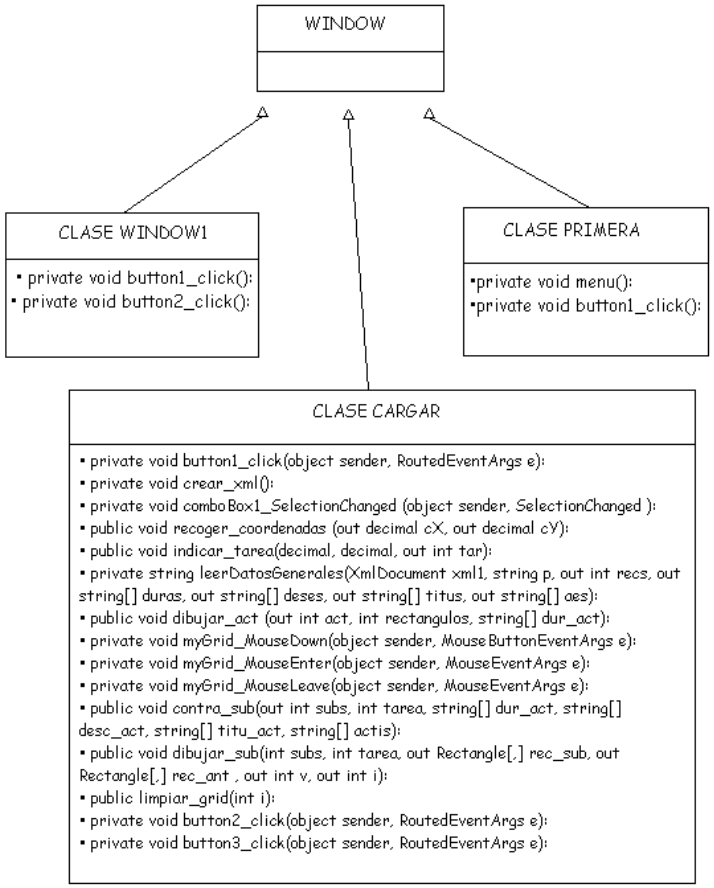


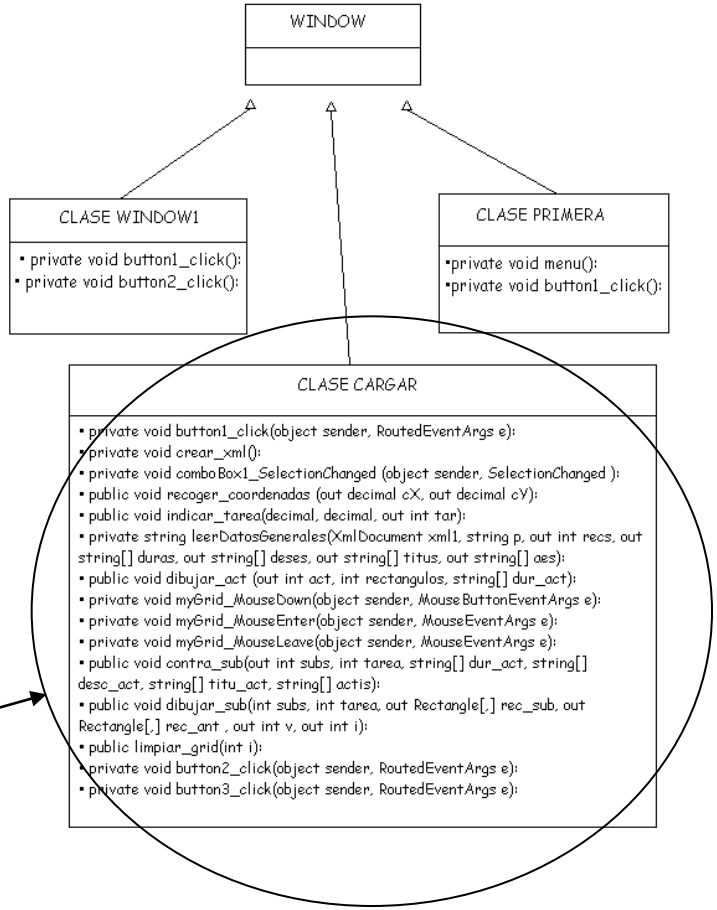
Diagrama de clases

- Todas las clases heredan de WINDOW.



Clases y métodos

- CLASE CARGAR
  - button1_click(...)
  - crear_xml()
  - combobox_SelectionChanged(...)
  - recoger_coordendas(...)
  - indicar_tarea(...)
  - leerDatosGenerales(...)
  - dibujar_act(...)
  - myGrid_MouseDown(...)
  - myGrid_MouseEnter(...)
  - myGrid_MouseLeave(...)
  - contar_sub(...)
  - dibujar_sub(...)
  - limpiar_grid(...)
  - button2_click(...)
  - button3_click(...)



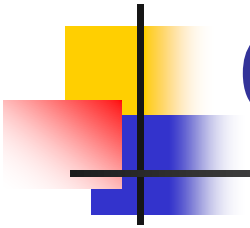
Estructura XML

- Tener en cuenta que:
  - Sólo se debe tener un nodo raíz.
  - Partes definidas por las etiquetas.

```
<?xml version="1.0" encoding="utf-8"
standalone="no"?>
<Procesos>
  <Proceso>
    <Id_proceso></Id_proceso>
    <Actividad>
      <Id_act></Id_act>
      <Duracion></Duracion>
      <Descripcion></Descripcion>
      <Titulo></Titulo>
      <Subactividades>
        <Subactividad>
        </Subactividad>
      </Subactividades>
    </Actividad>
  </Proceso>
</Procesos>
```

Código XAML de la interfaz

```
<Window x:Class="menu.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/prese
ntation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Principal" Height="298" Width="337"
Background="PeachPuff"
WindowStartupLocation="CenterScreen"
ResizeMode="NoResize">
    <Grid>
        <TextBlock FontSize="30"  Height="47"
         Margin="24,61,12,0" Name="textBlock1"
         VerticalAlignment="Top">Acceder al programa
        </TextBlock>
        <Button Margin="56,0,0,37" Name="button1"
         Height="27" VerticalAlignment="Bottom"
         Click="button1_Click"
         HorizontalAlignment="Left"
         Width="77">Aceptar
        </Button>
        <Button Height="27" HorizontalAlignment="Right"
        Margin="0,0,62,37" Name="button2"
        VerticalAlignment="Bottom" Width="83"
        Click="button2_Click">Cancelar
        </Button>
    </Grid>
</Window>
```



Conclusiones

- Conocimiento de un nuevo lenguaje: C#, WPF y XAML.
- División del trabajo.
- Ideas claras para programar.
- Idea global de cómo afrontar un trabajo.

Líneas futuras

- Extender funcionalidad para programar menús de navegación.
- Visualizar en formato *carrusel* todas las actividades cuando son un número grande.